**International Journal of Electronics,
Communication & Instrumentation Engineering
Research and Development (IJECIERD)
ISSN(P): 2249-684X; ISSN(E): 2249-7951
Vol. 4, Issue 1, Feb 2014, 87-96**

ERROR DETECTION AND ENHANCED DECODING OF DIFFERENT SET CODES FOR MEMORY APPLICATIONS

M V SUSHMA[1], C R S HANUMAN[2] & G LEENENDRA CHOWDARY[3]

[1]Research Scholar, Department of ECE, SITE, Tadepalligudem, Andhra Pradesh, India

[2]Associate Professor, Department of ECE, SITE, Tadepalligudem, Andhra Pradesh, India

[3]Assistant Professor, Department of ECE, SITE, Tadepalligudem, Andhra Pradesh, India

ABSTRACT

The key novel contribution of this paper is identifying and defining a new class of error-correcting codes whose redundancy makes the design of fault-secure detectors (FSD) particularly simple. Reliability, Availability and Serviceability are the three important parameters to be satisfied by any application. With further reduction in transistor size that leads to smaller dimensions, higher integration densities, and lower operating voltages, the reliability of memories is put into jeopardy. Single event upsets (SEUs) altering digital circuits are becoming a bigger concern for memory applications. Memory cells have been protected from soft errors for more than a decade; due to the increase in soft error rate in logic circuits, the encoder and decoder circuitry around the memory blocks have become susceptible to soft errors as well and must also be protected. A new approach to design fault-secure encoder and decoder circuitry for memory designs is introduced. The parity-check Matrix of an FSD-ECC (fault secure detector - error correcting code) has a particular structure that the decoder circuit, generated from the parity-check Matrix, is Fault-Secure. LDPC codes satisfies a new, restricted definition for ECCs which guarantees that the ECC codeword has an appropriate redundancy structure such that it can detect multiple errors occurring in both the stored codeword in memory and the surrounding circuitries.

KEYWORDS: Majority Fault Detection, Error Correction Codes, Difference Set Cyclic Codes

INTRODUCTION

A fault- tolerant nano scale memory architecture which tolerates transient faults both in the storage unit and in the supporting logic (i.e., encoder, decoder, corrector and detector circuitries) is introduced. The impact of technology scaling is affecting the reliability of memory applications not only in extreme radiation environments like spacecraft and avionics electronics, but also at normal terrestrial environments. Especially, SRAM memory failure rates are increasing significantly, therefore posing a major reliability concern for many applications.

Transient Faults

When a node in the system loses its effective charge due to ionized particle hit or various source of noises, it may cause the value of a node to be flipped in the circuit. However, the error does not permanently change the circuit, and it only generates a faulty bit value at the node that can last for one or few cycles. The transient fault rate is the probability that a single node loses its correct value during one clock cycle. A class of error-correcting codes (ECCs) that guarantees the existence of a simple fault-tolerant detector design is identified. This class satisfies a new, restricted definition for ECCs which guarantees that the ECC codeword has an appropriate redundancy structure such that it can detect multiple errors occurring in both the stored codeword in memory and the surrounding circuitries. The parity-check Matrix of an FSD-ECC has a particular structure that the decoder circuit, generated from the parity-check Matrix, is Fault-Secure.

Techniques to Mitigate Soft Errors

Soft errors are a major reliability concern for today's nanometer technologies. The errors in register files in Application Specific Integrated Circuits (ASIC) can quickly spread to various parts of the system and result in data corruption which may go unnoticed. Single Error Correction (SEC) Hamming code and Triple Modular Redundancy (TMR) provide a high-level mitigation solution for soft errors. By far, the most effective method of dealing with soft errors in memory components is by employing additional circuitry for error detection and/or correction.

The experimental results presented in this work for a 64-bit wide, 32-word entry register file, synthesized for IBM Cu-08 (90nm) standard cell ASIC technology, show that TMR is a better solution for latency sensitive ASIC applications. This technique increases the read access time by only 17% but incurs 204% area penalty. Whereas SEC applied on the 64-bit word size incurs 129% increase in read access time with area penalty only about 22%.

Technology scaling has made today's designs much more susceptible to soft errors. The ever decreasing supply voltages and nodal capacitances (required or constraining the power and making circuit transition faster) results in reduced critical charge (Qcrit) required to upset a node in digital circuits. The problem becomes more acute for aircraft and space electronics where high-energy neutrons at higher altitudes and heavy ions in space are more abundant.

Because of the prevailing predictions that soft error rate is increasing exponentially [1], there is a growing trend in the community to adopt soft error rate as a design parameter along with speed, area and power requirements.

The parity system allows for the detection of a soft error for a minimal cost in terms of circuit complexity and memory width (only a single bit is added to each word). The two disadvantages of this system are that the detected error cannot be corrected and if a double error has occurred then the check will not reveal that anything is wrong since the parity will match. This is true for any even number of errors. For example, if the data were stored with odd parity, the first error changes the odd parity to even parity (detectable error), but the second error changes the parity back to odd (non detectable error) [3]. In order to address these shortcomings, error detection and correction (EDAC) or error correction codes (ECC) is employed. Typically, error correction is achieved by adding extra bits to each data vector encoding the data so that the "information distance" between any two possible data vectors is, at least, three.

LITERATURE SURVEY

Desired ECC Properties

Among the ECC codes that meet the requirements of higher error correction capability and low decoding complexity necessary for mitigating soft errors, cyclic block codes have been identified as good candidates, due to their property of being majority logic (ML) decodable [8]. Unlike many other classes of codes LDPC codes are already equipped with very fast (probabilistic) encoding and decoding algorithms. It has been shown that these codes achieve a remarkable performance with iterative decoding that is very close to the Shannon limit. Consequently, these codes have become strong competitors to turbo codes for error control in many communication and digital storage systems where high reliability is required.

An LDPC code is defined as the null space of a parity check matrix **H** with the following structural properties: (1) each row consists of _ "ones"; (2) each column consists of "ones"; (3) the number of "ones" in common between any two columns, denoted _, is no greater than 1; (4) both _ and are small compared to the length of the code and the number of rows in **H** [1, 2]. Since _ and are small, **H** has a small density of "ones" and hence is a sparse matrix. For this reason, the code specified by **H** is called an LDPC code. The LDPC code defined above is known as a regular LDPC code. If not all

the columns or all the rows of the parity check matrix **H** have the same number of "ones" (or weights), an LDPC code is said to be irregular. Although LDPC codes have been shown to achieve outstanding performance, no analytic (algebraic or geometric) method has been found for constructing these codes. Gallager only provided a class of pseudo-random LDPC codes [1, 2]. Good LDPC codes that have been found are largely computer generated, especially long codes. Encoding of these long computer generated LDPC codes is quite complex due to the lack of code structure such as cyclic or quasi-cyclic structure. Furthermore, their minimum distances are either poor or hard to determine. LDPC codes can be classified into regular and irregular codes. With irregular codes improved performance is possible, since variable nodes with higher degrees (degree is number of adjacent nodes) collect more information from their adjacent check nodes and they get corrected first after a small number of iterations.

Difference Set Cyclic Codes (DSCCS)

DSCC is part of the LDPC codes, and, based on their attributes, they have the following properties:

- Ability to correct large number of errors;
- Sparse encoding, decoding and checking circuits synthesizable into simple hardware;
- Modular encoder and decoder blocks that allow an efficient hardware implementation;
- Systematic code structure for clean partition of information and code bits in the memory.

An important thing about the DSCC is that its systematical distribution allows the ML decoder to perform error detection in a simple way, using parity check sums.

Existent Majority Logic Decoding (MLD) Solutions

Plain ML Decoder

As described before, the ML decoder is a simple and powerful decoder, capable of correcting multiple random bit-flips depending on the number of parity check equations. It consists of four parts: 1) a cyclic shift register; 2) an XOR matrix; 3) a majority gate; and 4) an XOR for correcting the codeword bit under decoding. The input signal is initially stored into the cyclic shift register and shifted through all the taps. The intermediate values in each tap are then used to calculate the results {Bj} of the check sum equations from the XOR matrix. In the Nth cycle, the result has reached the final tap, producing the output signal (which is the decoded version of input) [1].

Plain MLD with Syndrome Fault Detector (SFD)

In order to improve the decoder performance, alternative designs may be used. One possibility is to add a fault detector by calculating the syndrome, so that only faulty code words are decoded [11]. Since most of the code words will be error-free, no further correction will be needed, and therefore performance will not be affected. Although the implementation of an SFD reduces the average latency of the decoding process, it also adds complexity to the design (see Figure 4). The SFD is an XOR matrix that calculates the syndrome based on the parity check matrix. Each parity bit results in a syndrome equation. Therefore, the complexity of the syndrome calculator increases with the size of the code. A faulty code word is detected when at least one of the syndrome bits is "1." This triggers the MLD to start the decoding, as explained before. On the other hand, if the codeword is error free, it is forwarded directly to the output, thus saving the correction cycles. In this way, the performance is improved in exchange of an additional module in the memory system: a matrix of XOR gates to resolve the parity check matrix, where each check bit results into a syndrome equation. This finally results in a quite complex module, with a large amount of additional hardware and power consumption in the system.

Proposed ML Detector/Decoder

In the proposed Majority Logic Detector/Decoder (MLDD), the data words are encoded using DSCC and then stored in the memory. When the memory is read, the code word is then fed through the ML detector/decoder before sent to the output for further processing. In this decoding process, the data word is corrected from all bit-flips that it might have suffered while being stored in the memory

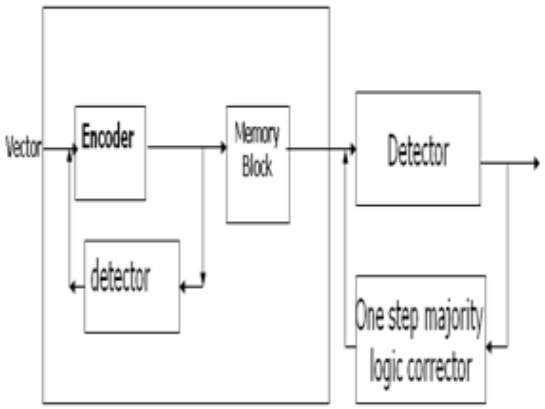

Figure 1: Main Architecture

HYPOTHESIS

The proposed technique is based on the following hypothesis: "Given a word read from a memory protected with DSCC codes, and affected by up to five bit-flips, all errors can be detected in only three decoding cycles."[1] This is a huge improvement over the simpler case, where N decoding cycles are needed to guarantee that errors are detected. The proof of this hypothesis is very complex from the mathematical point of view. Therefore, two alternatives can be used in order to prove it, which are given here. • Through simulation, in which exhaustive experiments have been conducted, to effectively verify that the hypothesis applies (see Section IV of reference [1]). • Through a simplified mathematical proof for the particular case of two bit-flips affecting a single word (see Appendix of reference [1]).

Let us assume that the hypothesis is true and that only three cycles are needed to detect all errors affecting up to five bits. The input signal is initially stored into the cyclic shift register and shifted through the taps. The intermediate values in each tap are then used to calculate the results {Bj} of the check sum equations from the XOR matrix. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all {Bj} is "0," the codeword is determined to be error-free and forwarded directly to the output. If the {Bj} contain in any of the three cycles at least a "1," the proposed method would continue the whole decoding process in order to eliminate the errors.

ELEMENT

The Muller **C-element**, or Muller C-gate, is a commonly used asynchronous logic component originally designed by David E. Muller. It applies logical operations on the inputs and has hysteresis. The output of the C-element reflects the inputs when the states of all inputs match. The output then remains in this state until the inputs all transition to the other state. This model can be extended to the Asymmetric C-element where some inputs only effect the operation in one of the transitions (positive or negative). The figure shows the gate-level and transistor-level implementations and symbol of the C-element.

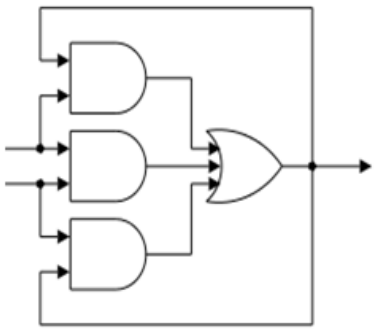

Figure 2: The symbol of the C-element

The C-element stores its previous state with two cross-coupled inverters, similar to an SRAM cell. One of the inverters is weaker than the rest of the circuit, so it can be overpowered by the pull-up and pull-down networks.

If both inputs are 0, then the pull-up network changes the latch's state, and the C-element outputs a 0. If both inputs are 1, then the pull-down network changes the latch's state, making the C-element output a 1. Otherwise, the input of the latch is not connected to either V_{dd} or ground, and so the weak inverter (drawn smaller in the diagram) dominates and the latch outputs its previous state.

ENCODER DESIGN

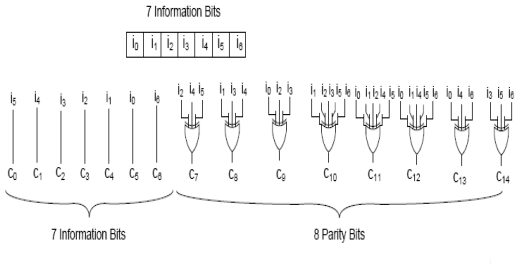

Figure 3: Encoder Design

The ABOVE shows the Basic Encoder which has information bits from i0 to i6, and these information bits are assigned to the remaining in a different pattern. Here C0 to C6 are considered as 7 Information bit and C7 to C14 as 8 parity bits. By considering the above Encoder design we construct the parity side of the Generation matrix that is from C7 to C14 as shown in the below Generation matrix.

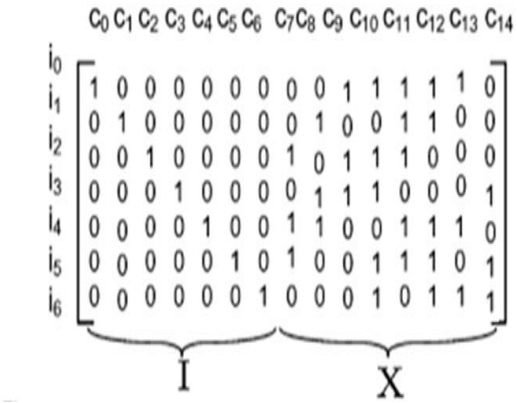

Figure 4: Generation Matrix for the (15, 7)

By calculating the Generation matrix G = IX we can get the Parity check matrix which is denoted by $H = X^T I$.

For detecting the errors which may occurred during transiting data from encoder to memory we are finding a syndrome matrix (S) to detect errors in encoder. Depend up on the density of the syndrome matrix we can find whether any error occurred or not, where $S = CH^{T.}$

In order to compute the syndrome for the code, we consider the following example of parity check matrix as shown:

In systematic form we have

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure 5: Parity Check Matrix H

H= Let r=$(c_0\ c_1\ c_2 c_3\ c_4\ c_5\ c_6)$ 1.1

be the received vector. Syndrome is given by

s = $(s_0\ s_1\ s_2\ s_3)$ = r.HT 1.2

Thus we have

$s_0 = c_0 + c_4 + c_5$ 1.3

$s_1 = c_1 + c_5 + c_6$ 1.4

$s_2 = c_2 + c_4 + c_5 + c_6$ 1.5

$s_3 = c_3 + c_4 + c_6$ 1.6

MLDC

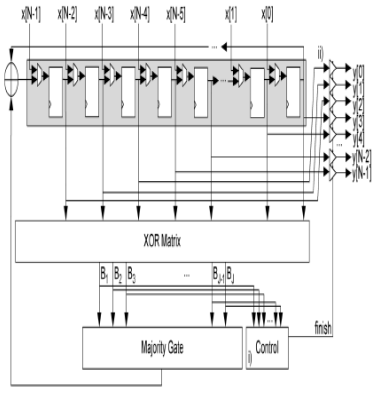

Figure 7: Schematic of MLDC

The ABOVE shows the basic ML decoder with an N-tap shift register, an XOR array to calculate the orthogonal parity check sums and a majority gate for deciding if the current bit under decoding needs to be inverted. The control unit triggers a finish flag when no errors are detected after the third cycle. The output tri state buffers are always in high impedance unless the control unit sends the finish signal so that the current values of the shift register are forwarded to the output y.

Detector for both the example codes mentioned before were coded in VHDL and simulated successfully. The tool Xilinx ISE 8.1iwas used for synthesizing the codes.

RESULTS

The plain Majority Logic Decoder, Majority Logic Detector / Decoder and Syndrome Fault

SIMULATION RESULTS

Plain Majority Logic Decoder

The input is provided to the cyclic shift register and after load signal goes high we find that 7 cycles and 21 cycles are needed to decode the code word irrespective of whether the code word is right or wrong for both the codes respectively. The „Complete‟ signal signifies when the process is complete and output is available.

MAJORITY LOGIC DECODER / DETECTOR

The input is provided to the cyclic shift register and after load signal goes high we find that only 3 cycles are needed to decode the code word if it is error free and 7 cycles and 21 cycles respectively are needed in case of an error. The „Complete‟ signal signifies when the process is complete and output is available

Detector Input

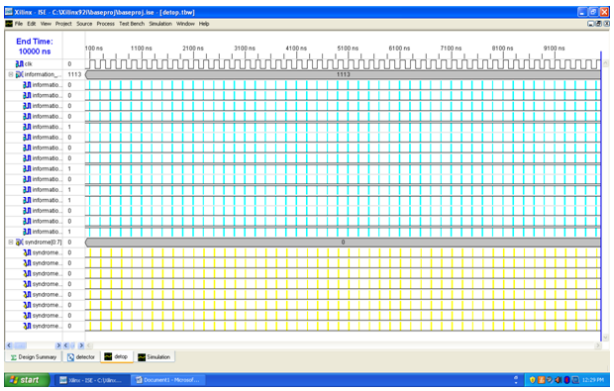

Figure 8

Detector Output

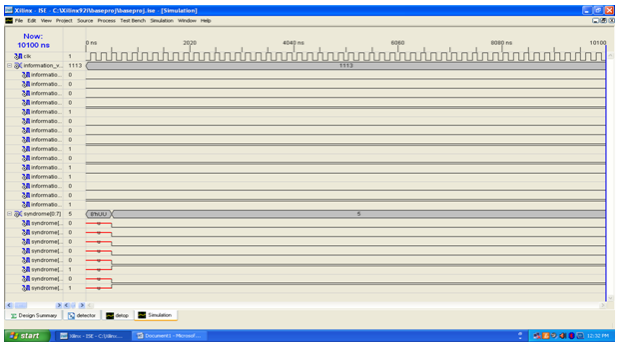

Figure 9

Synthesis Results

The previous subsection showed that the performance of the proposed design MLDD is much faster than the plain MLD version, but slightly lower than the design with syndrome calculator (SFD). The three designs have been synthesized

using Leonardo Spectrum for N values of 7 and 21 with scl05u library. The results are depicted in number of equivalent gates in Table 1.

Table 1: Synthesis Results of the Three Designs

	N=7	N=21
MLD	228	555
SFD	249	695
Overhead	9.2%	25.22%
MLDD	361	679
Overhead	58.33%	22.34%

The conclusions on the area results are given as follows: • The MLD design requires little area compared with the other two designs. However, as seen before, the performance results are not very good as all code words require N cycles for decoding. • The SFD version, which had the best performance, needs more area than the MLD does. Notice that the increment of area grows quicker than N does. • The MLDD version has a very similar performance to SFD, however it requires a much lower area overhead as the N value increases. The error detector in the MLDD has been designed to be independent of the size of the code, N. The opposite situation occurs, with the SFD technique, which uses syndrome calculation to perform error detection: its complexity grows quickly when the code size increases.

CONCLUSIONS

In this report, an efficient circuit to reduce soft error's in logic circuits is designed. The fault secure detectors are designed using a new class of error correcting codes called LDPC codes. Decoding time of the decoder is reduced using a min-sum algorithm for decoding. The encoder and decoder circuits around the memory are protected and LDPC code is proved as FSD-ECC. This improves the performance of the design with respect to the traditional MLD approach. The MLDD error detector module has been designed in a way that is independent of the code size. This makes its area overhead quite reduced compared with other traditional approaches such as the syndrome calculation (SFD). The ML decoder circuitry is used here as a fault detector so that read operations are accelerated with almost no additional hardware cost. By using the C-elements we can reduce the power consumption by its unique property. The results show that the properties of DSCC-LDPC enable efficient fault detection.

REFERENCES

1. Shih-Fu Liu, Pedro Reviriego, and Juan Antonio Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications" , IEEE Transactions on Very Large Scale Integration (VLSI) systems, vol. 20, no. 1, January 2012.
2. C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," IEEE Trans. Device Mater. Reliabil, vol. 5, no. 3, pp. 397–404, Sep. 2005.
3. R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," IEEE Trans. Device Mater. Reliabil., vol. 5, no.3, pp. 301–316, Sep. 2005.
4. J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," Automata Studies, pp. 43–98, 1956.
5. M. A. Bajura et al., "Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs," IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 935–945, Aug. 2007.

6. R. Naseer and J. Draper, “DEC ECC design to improve memory reliability in sub-100 nm technologies,” in Proc. IEEE ICECS, 2008, pp.586–589.
7. S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.
8. S. Reed, “A class of multiple-error-correcting codes and the decoding scheme,” IRE Trans. Inf. Theory, vol. IT-4, pp. 38–49, 1954.
9. J. L. Massey, Threshold Decoding. Cambridge, MA: MIT Press, 1963.